VZDU # 34 JAN/FEB 1992

VZ DOWN UNDER CLUB

Editor
Harry Huggins
12 Thomas Str.
Mitcham. 3132
03-873-1408

Treasurer
Ron Allen
2 Orlando Str.
Hampton. 3188
03-598-4534

HAPPY NEW YEAR EVERYONE.

What has the NEW YEAR in store for VEE ZEDers? A lot of good programs and suggestions, and just in case I have'nt mentioned it before ------a lot of contributions to the Newsletter. Remember I expect (hope) one contributation per member per year (per haps)! So what about it.

On behalf of the club members I extend to Bob Kitch our special thanks for the support he has given the VZ clubs in the past, and we hope for it to continue. He is a very regular contributor, and without his support the VZDU would have folded long ago. Thanks Bob from me and all our club members

I understood that there was going to be one more instalment of the series on Writing Mystry games by David Wood. Perhaps I was mistaken. Thanks David for a very good series. I know that many followed it by the requests I got for the Demo program on disk/tape. However David specified it was not to go out on disk/tape. I have had 4 requests for the printout (that may be borrowed from our library). David has moved onto bigger and better (?) computers. If it is IBM Compat. I hope to hear from him on IBM.

Paul has not come up with a Games Column, but I have received a Review of 3 games from Tim Pendlebury, which he wrote whilst preparing to break up the Watson/McLean monopoly of the High Scores.

Many of our members have expanded to IBM Compatiable computers. I would like to hear from ex VZers, and maybe form another group in the IBM field. I don't aim to start another Newsletter, but we could exchange memories of when things were simple with a VZ.

You will find the 2 items missed from last issue. The circuit for Ben's remote control, and listing 5 from Bob's program. Sorry about that. You will have noticed we had some printing problems.

Harry

READ 00190

BUNNY

You can rest easy. This is not an entry in the competition. It was comverted from another computer.

I shall go through it and explain how it works. If you have already worked it out, then there is no need to read this.

```
00005 CLS:                    BUNNY
00010 LPRINT "
00030 LPRINT:LPRINT:LPRINT
```

To here you should need no explanation.

00120 FOR I=0TO4: READ B(I): NEXTI

This is a simple READ instruction loop that reads the first 5 items of DATA. That is B(0)=2: B(1)=21: B(2)=14: B(3)=14: b(4)=25

00130 GOSUB260

260 FORI=1TO6: LPRINTCHR$(10);: NEXT;I------CHR$10 is LINEFEED. So this put in 6 blank lines and returns to run 140.

00140 L=64----No comment here now.

00160 LPRINT-----This is another blank line.

00170 READ X:IFX<0THEN160

READ X. Earlier we read B. It matters not what you call the data you read, it will be the next item from DATA. In this case it will be 1. X=1.It is not <0 nor >128 so we go to 180.

00175 IFX>128THEN240

00180 LPRINT TAB(X);:READY --We will lprint at TAB(1)--But first we READ Y. Y will be 2.

00190 FORI=XTOY:J=I-5*INT(I/5)

To get the value of J enter this formula in your computer and enter the values of X and Y.

FOR I=X to Y. That is 1 to 2.

First J=I-5*INT(I/5). It will be 1

00200 LPRINTCHR$(L+B(J));

Now the tricky part. What are we going to lprint? CHR$(L+B(1)) B(1) is 21. L is 64. So we print CHR$85. 85 is U. NEXT goes back for another I.

00210 NEXT I

J=I-5*INT(I/5)

This time J=2. B(2) is 14. 64+14=78. CHR$(78) is N. So it lprints UN.

00220 GOTO 170 Back to 170

00170 READX:IFX<0THEN160

READ X.

X=-1. This time it is less than 0 so we goto 160.

00160 LPRINT.: That is a line feed. So we have got the TAB(and the line feed by reading DATA. And so far we have had UN printed. After 160 is 170.

00170 READX:IFX<0THEN160

READ X. This time X=0.

so we go to 180.

00175 IFX>128THEN240

00180 LPRINT TAB(X);:READ Y --We will lprint at TAB(0)--But first we

READ Y. Y will be 2.
00190 FORI=XTOY:J=I-5*INT(I/5)

FOR I=X to Y. That is 0 to 2.First J=I-5*INT(I/5).
This time there are 3 numbers (from 0 to 2) and the formula gives us o,1,2.
00200 LPRINTCHR$(L+B(J));
This time it prints out 64 plus B(0),B(1) and B(2). That is BUN
NEXT goes back for another I.
00210 NEXT I

00170 READX:IFX<0THEN160

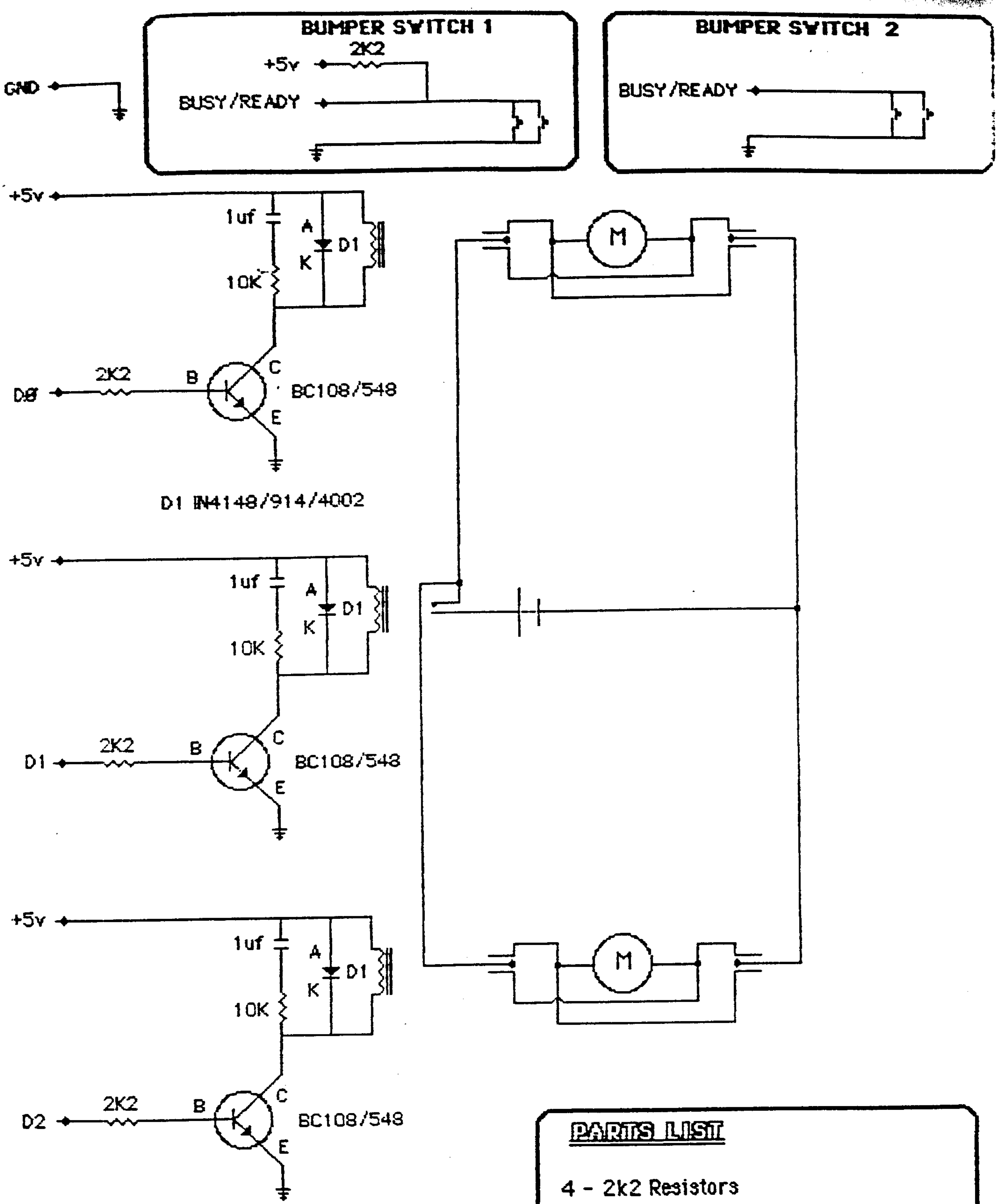
READ X. This time X=45
so we go to 180.
00175 IFX>128THEN240
00180 LPRINT TAB(X);:READ Y --We will lprint at TAB(45)--But first we READ Y. Y will be 50.
Notice that this time there is no DATA -1 (line feed) so this will be printed on the same line as BUN, but at TAB(45)
00190 FORI=XTOY:J=I-5*INT(I/5)

FOR I=X to Y. That is 45 to 50.First J=I-5*INT(I/5).
This time there are 6 numbers (from 45 to 50) and the formula gives us o,1,2,3,4,0
00200 LPRINTCHR$(L+B(J));
This time it prints out 64 plus B(0),B(1), B(2), B(3), B(4) and B(0) That is BUNNYB NEXT goes back for another I.
00210 NEXT I
And so it will go on. All controlled by READing X and Y from the DATA. At the end of this is a little program you should type in and alter the values of X and Y and it will print out the sequence of values for B(j) that will print. Even the END is called from DATA. The last DATA bit is 4096.Line 175. If X>128 then240. Line 240 is GOSUB 260. 260 puts 6 blank lines in and returns to 240 and is sent to line 450 which is END.

```
00240 GOSUB260:GOTO450
00260 FORI=1TO6: LPRINTCHR$(10);: NEXT I
00270 RETURN
00290 DATA2,21,14,14,25
00300 DATA1,2,-1,0,2,45,50,-1,0,5,43,52,-1,0,7,41,52,-1
00310 DATA1,9,3[illegible],50,-1,2,11,36,50,-1,3,13,34,49,-1,4,14,32,48,-1
00320 DATA5,15,[illegible]1,47,-1,6,16,30,45,-1,7,17,29,44,-1,8,19,28,43,-1
00330 DATA9,20,27,41,-1,10,21,26,40,-1,11,22,25,38,-1,12,22,24,36
00335 DATA-1
00340 DATA13,34,-1,14,33,-1,15,31,-1,17,29,-1,18,27,-1
00350 DATA19,26,-1,16,28,-1,13,30,-1,11,31,-1,10,32,-1
00360 DATA8,33,-1,7,34,-1,6,13,16,34,-1,5,12,16,35,-1
00370 DATA4,12,16,35,-1,3,12,15,35,-1,2,35,-1,1,35,-1
00380 DATA2,34,-1,3,34,-1,4,33,-1,6,33,-1,10,32,34,34,-1
00390 DATA14,17,19,25,28,31,35,35,-1,15,19,23,30,36,36,-1
00400 DATA14,18,21,21,24,30,37,37,-1,13,18,23,29,33,38,-1
00410 DATA12,29,31,33,-1,11,13,17,17,19,19,22,22,24,31,-1
00420 DATA10,11,17,18,22,22,24,24,29,29,-1
00430 DATA22,23,26,29,-1,27,29,-1,28,29,-1,4096
00450 END
```

LISTING 5

```
5    '          *************************
10   '          ***  1536 VZ SCREENS  ***
20   '          ***    VERSION 1.3    ***
30   '          ***  R.B.K.  18/5/86  ***
40   '          *************************
50   '
88   '
89   '***FIND TOP OF MEMORY AND PROTECT 20 BYTES BY LOWERING TOM.
90 IF PEEK (30791)-77 THEN GOTO 500          :'7847H AS LOAD CHECK.
100 M1-PEEK(30898):L1-PEEK(30897)            :'TOM POINTERS.
110 TM-M1*256+L1-21                          :'PROTECT 20 BYTES.
120 MS-INT(TM/256):LS-TM-MS*256              :'NEW TOM POINTERS.
130 POKE 30898,MS:POKE 30897,LS              :'SET NEW TOM.
140 CLEAR 50                                 :'RESET ALL POINTERS.
150 'THIS ACTION DESTROYS THE VLT-ALL VARIABLES MUST BE RENEWED.
160 M1=PEEK(30898):L1=PEEK(30897)            :'NEW TOM POINTERS.
170 ST=M1*256+L1+1                           :'NEXT ADDR IN RESERVED
180 MS=INT(ST/256):LS=ST-256*MS              :'START OF ROUTINE.
190 IF ST>32767 THEN ST=ST-65536             :'CONVERT TO SIGNED INT
198 '
199 '***LOAD MACHINE CODE INTO PROTECTED TOM AND SET USR POINTER
200 CLS:PRINT@230,"LOADING MACHINE CODE"
210 CS%=0                                    :'CHECK SUM COUNTER.
220 FOR ID=ST TO ST+15                       :'LOAD 16 BYTES TO TOM.
230    READ VL%:POKE ID,VL%:CS%=CS%+VL%      :'LOAD M/L INTO PROTECT
240 NEXT ID
250 IF CS%<>1483 THEN GOTO 1000              :'WRONG DATA.
260 POKE 30863,MS:POKE 30862,LS              :'SET USR POINTERS.
270 POKE 30791,77                            :'SET LOAD CHECK.
399 '
400 '***Z-80 BLOCK MOVE ROUTINE.
410 DATA 058,033,121   :'LD A,(7921H) PICK UP VALUE FROM USR().
420 DATA 033,000,112   :'LD HL,7000H  (#28672D START VIDEO RAM)
430 DATA 017,001,112   :'LD DE,7001H  (#28673D NEXT)
440 DATA 001,255,007   :'LD BC,07FFH  (#2047D  SIZE OF VIDEO RAM)
450 DATA 119           :'LD (HL),A
460 DATA 237,176       :'LDIR         (BLOCK LOAD COMMAND)
470 DATA 201           :'RET
498 '
499 '***INITIALIZE DELAYS - CONTROL SPEED OF EXECUTION BY D.
500 T%=0                                     :'TONE IS REST (0-31)
510 D%=4                                     :'DELAY IS SHORT(1-9)
520 P%=30744                                 :'SET UP SCREEN MODE.
594 '
595 '***THIS LOOP DOES A LOT OF WORK.
596 '***IT LOADS ALL VALUES INTO THE A-REG FOR VIEWING.
597 '***IT SWAPS BETWEEN HI- AND LO-RES SCREEN MODES.
598 '***IT SWITCHES BETWEEN NORMAL AND INVERSE SCREEN TYPES.
599 '***IT SWITCHES BETWEEN GREEN AND ORANGE BACKGROUNDS.
600 FOR A%=0 TO 255             :'ALL VALUES PUT INTO A=REG.
610    MODE(0):POKE P%,0
620    CLS:PRINT@234,"VALUE = ";A%:SOUND T%,D%
630    X=USR(A%)
640    COLOR,0:SOUND T%,D%
650    COLOR,1:SOUND T%,D%
660    POKE P%,1
670    X=USR(A%)
680    COLOR,0:SOUND T%,D%
690    COLOR,1:SOUND T%,D%
700 '  HI-RES SCREENS.
710    MODE(1)
720    X=USR(A%)
730    COLOR,0:SOUND T%,D%
740    COLOR,1:SOUND T%,D%
750 NEXT A%
800 GOTO 600
999 '***WRONG DATA MESSAGE.
1000 CLS:PRINT@230,"ERROR IN DATA ";CS%
```



PARTS LIST

4 - 2k2 Resistors
3 - 10k Resistors
3 - 1uf Capacitors
2 - Double Pole Double Throw 5v Relays
1 - Single Pole Single Throw 5v Relay
3 - BC108 or similar transistors
3 - IN4001 or similar Diodes
4 - N.O. Microswitches
2 - Motors
1 - Battery to Suit motors

mputer Controlled Robot
awn by Ben Hobson
1-08-1991
r use on Parallel Printer Port

FAST BASIC

Connecting Machine Code to Basic

by Bob Kitch

I have written about my FAST BASIC technique and used it in my LIVENUP and SOUND series of programs for the VZ. It is a hybrid language. A number of users have also asked how it is done. I will endeavour to provide an interesting and illuminating discussion of the technique.

This is an invitation to explore this significant enhancement of normal Basic. For programmers who are looking for better (smarter or faster) ways to write programs, then this is of interest. Also, for Basic programmers who are wrestling with Z80 Machine Code, then FAST BASIC provides an ideal introduction.

To fully understand FAST BASIC we need to explore -

1. how a Basic program is structured in memory,
2. how to reserve an area of memory for Machine Code, and
3. understand how to connect Machine Code with a Basic program.

Before I commence, it is worth mentioning some books, that most users will have in their collections and that will reinforce the explanations that I provide. The DSE Technical Manuals for the VZ 200 and 300 (1983 and 1985) are useful. Steve Olney's book (1987) on Assembly Language for Beginners is also very useful on these topics. For a well-paced introduction to Z80 Assembly Language programming, the two Tandy/Radio Shack books by Bill Barden (1979 and 1982) for the TRS-80 are as good as any, particularly since the ROM in the TRS-80 and VZ is very similar. You may have to hunt around for copies of these books as they are out of print.

Let's start by looking at the structure of a Basic program in memory.

1. FIVE BASIC TABLES & MEMORY UTILIZATION.

By knowing the organization of a program in memory, we can make it perform more efficiently. Although you have probably never thought of it this way, (and it is certainly not mentioned in the VZ Manuals!), a Basic program running on the VZ actually consists of FIVE TABLES. Basic programmers are very familiar with the first of these, the PROGRAM STATEMENT TABLE (PST), as this is the actual tokenized (compressed) program, or source, that is written.

The other tables are "written" by the Interpreter when the program is executed. The VARIABLE LIST TABLE (VLT) actually consists of two parts, the SIMPLE VARIABLE TABLE (SVT) and the DIMENSIONED VARIABLE TABLE (DVT). These are positioned above the PST and "grow" as the program executes and "discovers" more variables - that is they are dynamic whereas the PST is static once execution commences. (The PST grows as you write the program).

The remaining two tables are located below the Top-Of-Memory (TOM). (For DOS systems, the TOM is located below the DOS Vector) The

uppermost one is the STRING AREA that stores the string variables used by a Basic program. Note that quoted strings are stored in the PST. The number of bytes reserved is fixed by the CLEAR command and the default is 50 bytes. It is a static table.

Below the String Area is the STACK which "grows" downwards in memory and is dynamic or changeable. The Stack is used by the Z80 for PUSHing and POPping registers and also by the Basic program to keep track of GOSUB/RET and FOR/NEXT calls.

This completes the five tables used by Basic. The intervening area of RAM below the Stack and above the VLT is called FREE SPACE LIST (FSL) or unused memory. It is bounded above and below by dynamic tables (Stack and VLT) and the size of FSL changes as program execution proceeds.

(You may want to read the preceding paragraphs a few times, as it gives a rather "different" view of a Basic program to what you are probably used to.)

A memory map of this situation with the <pointers> indicating the start and end of the various tables would be as follows -

```
  !============================! <IY reg>
  ! Top of Physical Memory     !
  ! DOS Vector on disk systems!
B !============================! <78B1/2H>
A ! String Area                !
S !----------------------------! <78A0/1H>
I ! Stack                      !
C !----------------------------! <78E8/9H>
  !                            !
W ! Free Space List            !
O !                            !
R !----------------------------! <78FD/EH>
K ! Dim'd Variable Table       !
  !----------------------------! <78FB/CH>
A ! Simple Variable Table      !
R !----------------------------! <78F9/AH>
E ! Program Statement Table    !
A !============================! <78A4/5H>
  ! Communication  Area        !
  !----------------------------! 7800H
  ! Video RAM                  !
  !----------------------------! 7000H
  !                            !
```

All of these subdivisions of RAM used by Basic are not fixed and move up and down depending upon what actions are performed. Inserting or deleting a line from a Basic program alters the size of the PST. Similarly, defining a new variable, increases the length of the VLT. The String Area can only be changed by the CLEAR statement and this causes the relocation of the Stack. This is a very drastic action and results in a major "reset" of the various tables. How does the Interpreter keep track of these tables?

Since the origin of the tables may shift, their addresses are kept as POINTERS in the Communications Area of the VZ. I trust that there is beginning, some dawning of the relevance of information given in the VZ Technical Manuals. On page 11 or 20, of the VZ 200 or 300 manual respectively, the various pointers are given. Olney page 125 and 129-130 provides a clear description of the foregoing. If you have these publications, then spend some time understanding the structure of a Basic program in memory.

The manner in which information is packed in each table is a very interesting topic, but will be left for another time as it is not required at present.

2. RESERVING RAM FOR MACHINE CODE.

I will discuss the various ways that Machine Code (M/C) can be located in memory along with the Basic program. They fall into 4 types -

i. Loaded into RAM with the Basic program - appended or embedded.
ii. Poked into RAM from the Basic program - set-up time required.
iii. Already existing in ROM - ready to run.
iv. Loaded into RAM as seperate programs - two or more modules.

Type i. can be located below the PST and above the Communication Area. The COPYPRO program of Larry Taylor uses this technique. Method 1 in the VZ Technical Manuals describe the method. Method 2 in the Technical Manuals describes how to locate M/C above the PST and below the VLT. Olney on page 47 also describes this technique. The latter technique is a better method than loading M/C below the PST. Both methods have the advantage that a single module, of combined M/C and Basic, is loaded into co-joined memory locations.

Type ii. methods have a number of variants. Generally the M/C is held in DATA statements within the Basic program located in the PST. The thing that is interesting about this technique is that the M/C can be POKEd into the Basic Program and embedded in the PST, POKEd into FSL or reserved memory, or put into the VLT or String Area. It is obviously very flexible. Techniques that modify the PST imply that the POKEing of the M/C need only be done once and that it is subsequently loaded along with the Basic program. The other techniques are not particularly memory efficient, as two copies of the M/C are held in the PST, in the DATA statements, and also in its run-time location. Method 3 in the VZ Technical Manuals lowers the Top-Of-Memory to create a reserved are of RAM below the DOS vector and above the String Area. Olney also describes the method on pages 37-38. Following this lowering it is necessary to reset all of the Basic pointers with a CLEAR. (now you understand why!). This is by far and wide the best method to locate M/C as it gives it an unambiguous and protected area in which to operate. It is easy to debug also as the area can be disassembled if required. Once a reserved area of memory is made, the M/C can be POKEd in as is done in my TONEGEN program, or it may be loaded in from disk as is done with MOVEUP in my LIVENUP program. The memory map for this arrangement is as follows -

```
!============================! <IY reg>
! Top of Physical Memory     !
```

```
! DOS Vector on disk systems!
!===========================!
!                           !
! Reserved TOM Area         !
!                           ! <788E/FH>
!===========================! <78B1/2H>
! String Area               !
!---------------------------! <78A0/1H>
! Stack                     !
!---------------------------! <78E8/9H>
!                           !
! Free Space List           !
!                           !
!                           !
```

There are other methods of POKEing M/C into RAM. The one used in my SOUND EFFECTS program, simply POKEs the code into FSL, but this can be risky because the VLT or the Stack could "grow" into the M/C and corrupt it. This can be a frustrating bug to discover. Use it only when you have a short program and lots of memory!

On pages 38 to 47 of Olney there are a couple of other Type ii. methods described. They have certain restrictions that make them difficult to use and require that short pieces of M/C be used. Their advantage is that once loaded, they are embedded in the Basic program and become Type i. The code is read from DATA statements and POKEd into REM statements or string variables located within the PST. The sections of the PST so altered, cannot be editted after this modification. These techniques do have the advantage of only requiring a single program to be loaded. Barden (1982) devotes Chapter 5 to these techniques also.

There is a further Type ii. technique that has had a whole book written about it! Lewis Rosenfelder's (1983) "Magic Memory" techniques simply read the M/C from the PST and place it into arrays stored in the DVT - no poking is required. Very long M/C programs can be stored in this manner. As the DVT is pushed up by the SVT, it is necessary to use the VARPTR command to keep track of the start of the M/C.

Type iii. are very satisfying to use and simple to set-up. You are using M/C that already exists within the resources of the ROM. There are a lot of useful routines contained in the VZ's ROM. A good knowledge of the ROM is required to use this technique. My SOUND EFFECTS program uses this technique to use the BEEP routine located in ROM at 3450H. The VZ Technical Manuals contain outlines of the "useful" ROM routines.

Type iv. techniques are best used by programmers writing for DOS-based machines and are "two module" approaches to the problem. Two (or more) programs are loaded from disk. The first will usually be the Basic program that loads into the PST. Upon execution, it lowers the Top-Of-Memory to make a reserved area, does a CLEAR to reset pointers, and then BLOADs a M/C program into the reserved area. The Usr Pointers are then set to the entry point of the M/C. Somewhere in the Basic program the USR command is executed and the M/C run. Upon RETurn, the Basic program continues. My Basic THROWUP program repeatedly calls MOVEUP, located in reserved TOM position, to achieve the screen moves

and is a Type iv. method. The M/C is usually written in Assembly Language using the EDASM, rather than POKEing decimal values.

3. THE USR() STATEMENT- using Machine Code from Basic.

The usual method of establishing a link between M/C and a Basic program is to call the USR command. This command is very convenient and incredibly powerful but is almost ignored in the VZ programming manuals. My REAL TIME CLOCK program is connected by stealing the Interrupt Vector - a totally different technique. M/C is used to speed up certain procedures that are too slow in Basic. Olney (1987, page 33) provides an interesting discussion of his views on the relationship between Basic and M/C. Nothing beats the Basic language for a quick and simple way to program applications. But, when super-fast execution speed and truly economical memory usage is required then M/C is supreme. This is in fact the essence of FAST BASIC.

In order to implement the USR command, a Basic program and some M/C must be in memory. The previous section discussed various methods and places where M/C can be placed in RAM. M/C in ROM may also be linked via the USR statement. The Usr Vector or Pointer must be initialized so that the Z80 knows where to continue code execution. Furthermore, the M/C routine must end with a RETurn command so that an orderly resumption of Basic takes place. Additionally, it may be desirable to pass parameters between the Basic program and M/C.

The USR pointer is located at 788E/FH (30862/3D) in the Communication Area. When the VZ boots up, this vector is initialized to 4A1EH and executes an error message. (?FUNCTION CODE ERROR IN XX) This prevents the use of the USR statement without first setting the pointer.

For those that are interested, the Basic Interpreter at 24FFH tests for the USR token (C1H) and, if found, jumps to the verb action routine located at 27FEH. You may wish to Disassemble the routine.

When the USR pointer is set to the address of the M/C routine it can be called by using X=USR(X). In the simplest case the argument X is a dummy, that is, it is not otherwise used in the program. The statement causes a jump to the memory address indicated by the Usr Pointer, executes the M/C routine, and then returns to the following Basic statement. In a way, it resembles the Basic GOSUB statement. Thus the link between Basic and M/C is made.

In other situations a parameter can be exchanged between the Basic and M/C program, which can be defined as constants of all types of variables and also strings. The parameter being passed is stored in the Basic accumulator (WRA1) on executing the USR command. If a string variable or constant is used, then WRA1 acts as a memory pointer to the string descriptor block (length and address). On RETurn to Basic from the M/C routine, the content of WRA1 will be stored in the variable defined by the USR(X) and can be used by the Basic program.

If the previous paragraph was not clear to you and you only wish to pass integer values between programs, then this method is easier to understand. The integer constant or variable enclosed in the ( ) is passed to locations 7921/2H (31009/10D) and can be accessed by the M/C

routine. An example of the use of this facility was given in APC Feb. 1986, page 127. - although a considerably more elegant and shorter method is available.

Well there was some pretty heavy going in that description, but I hope you will try to work through it, as it provides a very good insight into the Basic Interpreter in the VZ. Any queries are always welcome via the Editor. I will try to answer them and provide some programs that demonstrate the various techniques if there is sufficient feed-back. FAST BASIC is fun and it permits incredible things to be done with the VZ.

```
*****************
*               *
*   PROGRAM-1   *
*               *
*****************
```

```
10 POKE 30862,80:POKE 30863,52 :'SET USR POINTERS TO BEEP
20 A%=250:X=0
30 X=USR(A%)
40 PRINT PEEK(31009);PEEK(31010);A%;X
50 A$=""
60 A$=USR("AFTER")
70 PRINT A$
80 STOP
```

THREE REVIEWS

by TIM PENDLEBURY

REVIEW 1: <u>DEFENCE PENETRATOR</u>

"Defence Penetrator" is th first game Tom Thiel wrote, of the three games I will review.

Defence Penetrator has six (6) stages, the first being the easiest to complete and the last two (2) the hardest. The game is made harder by having to shoot the ammo and fuel boxes while avoiding the missiles and fireballs. You are always moving so you can't come to a complete halt. Try not to go into the valleys as you have to slow down and it's a waste of time.

I have seen this game, and "Road Warrior", using joysticks, but only on disk, too long for tape (5 minutes I think) and it doesn't always load.

Movement keys are—Q = UP, A = DOWN, - = FORWARD, 0 = BACKWARD, M = FIRE, 3 = PAUSE, 4 = UNPAUSE.

I really enjoyed this game.

REVIEW 2: <u>ROAD WARRIOR</u>

I played "Road Warrior" some weeks earlier under the name "Pursuit". This game is apparently the first version of "Morgoth".

"Road Warrior" is a maze with no way out. It has no stages and a scoring system which disappears too quickly to be recorded. As you make your way around the maze you have to avoid the objects that I call "shadows". They are the same colour as the background and you can only see their outlines. You have two (2) "screens" on your viewer and you have to constantly watch both screens. The lefthand and the largest of the screens shows the maze, your white ship, the "shadows", and little "F"s which I think mean "Fuel". The righthand screen is a radar. It shows your ship and the "shadows" as white and black dots. Your ship is always moving fast and you have little reaction time, making the game hard.

Movement keys are—Q = UP, A = DOWN, - = RIGHT, 0 = LEFT.

I found this game to be interesting, but frustrating.

REVIEW 3: <u>MORGOTH</u>

"Morgoth" is an update version of "Road Warrior" and has similar problems.

Like "Road Warrior", "Morgoth" has a maze with no way out. Your little ship has now grown up and now when you release the buttons it stops. You still have to avoid the "shadows". Having graphic flicker running across my screen makes it harder to see them. As you move around the maze you collect different objects each having its own score. The scoring system has improved, but it also disappears too quickly.

Movement keys are: Q = UP, A = DOWN, - = RIGHT, 0 = LEFT, 3 = PAUSE, 4 = UNPAUSE.

I enjoyed "Morgoth" much better than I did "Road Warrior" and I think that to get to the harder levels you must collect all the different objects.

THE HIGH SCORES

GAME	SCORE	LEVEL	HOLDER
DAWN PATROL	78100		Paul Frantz
CRASH	881		Matthew McLean
DIG OUT	52500		Kenley McLean
HAMBURGER SAM	51000		Roger McLean
LADDER CHALLENGE	23970		Peter Watson
KAMIKAZE	113410		Peter Watson
TEN PIN BOWLS	215		Mitch Pendlebury
VZ INVADERS	30160		Peter Watson
GALAXON	328,460		Mathew McLean
PENGUIN	2800		Matthew McLean
LUNAR LANDER	52520		Peter Watson
SUPER SNAKE	1918		Peter Watson
MAZE OF ARGON	78306		Peter Watson
ASTEROIDS	110000		Peter McLean
CIRCUS	3180		Matthew McLean
PANIK	11090		Martin Wedgwood
HOPPY	25550		Matthew McLean
GHOST HUNTER	23400		Chris McLean
KNIGHTS & DRAGONS	5300	Easy	Peter Watson
KNIGHTS & DRAGONS	1200	Expert	Peter Watson
SPACE RAM	1441		Matthew McLean
MISSILE ATTACK	52000		Heru McLean
BUST OUT	2600		Peter Watson
PLANET PATROL	1091		Peter Watson
DEFENCE PEN.	1563		Peter Watson
PHAROAH'S CURSE	135 g/bars	Skill5	Peter Watson
STAR BLASTER	787	level 1	Tim Pendelbury
STAR BLASTER	683	level 2	Tim Pendelbury
STAR BLASTER	625	level 3	Tim Pendelbury
STAR BLASTER	419	level 4	Tim Pendelbury
STAR BLASTER	219	level 5	Tim Pendelbury

STOP PRESS

Paul has advised us that due to changed circumstances he can no longer write our GAMES COLUMN.

We extend our thanks to Paul for past support and help he has extended to the club, and wish him success in his new undertaking. (Ed. HH.)

TRADING POST

EPROMS for EXTENDED DOS. and BASIC

Are available from

Bob Kitch
7 Eurella Str.
KENMORE Q'ld. 4069

FOR SALE by various members :-

VZ programming by Tim Hartnell. $3.50
VZ OMNIBUS by Tim Hartnell $5.00
VZ Giant Book of Games by Tim Hartnett $5.00
VZ 300 computer with plug pack and manuals $45.
VZ 16K memory expansion. $30.
VZ Cassette recorder. $20.
~~VZ Printer Plotter. $50.~~
NEC keyboard, adaptable to VZ 200 or 300. $20.
Printer SEIKOSHA GP250X as in last issue.
Keyboard Ex Totalisator. Would be adaptable. $15.

There are several sellers, so get in touch with me. If you wish to make an offer I will be pleased to pass it on.(Ed.)

OTHER V Z USER GROUPS

H.V.V.Z.U.G
P.O.Box 161
JESMOND NSW.2299.

DISKMAG
P.O.Box 600.
Taree NSW. 2430.

CENT.VIC.COMP.Club
24 Breen St.
BENDIGO VIC 3550

BRISBANE VZUG
63 Tingalpa St.
WYNUM West. Q'ld. 4178

Graeme Bywater
P.O.Box 388
MORLEY W.A. 6062